**PHASE 4 PROJECT SUBMISSION**

**PROJECT 6 - CUSTOMER CHURN PREDICTION**

**TEAM MEMBERS:**

1. Rithees S M - 2021504032
2. Akshay G S – 2021504502
3. Dhanesh C N E – 2021504510
4. Divyavarshini K – 2021504513

**Problem Definition:**

The project involves using IBM Cognos to predict customer churn and identify factors influencing customer retention. The goal is to help businesses reduce customer attrition by understanding the patterns and reasons behind customers leaving. This project includes defining analysis objectives, collecting customer data, designing relevant visualizations in IBM Cognos, and building a predictive model.

**Phase Objective:**

In this technology projects you will continue building your project by performing different analysis, model building and evaluation as per the project requirement. Perform different analysis and visualization using IBM Cognos.

**Dataset Link:**

[**https://www.kaggle.com/datasets/blastchar/telco-customer-churn**](https://www.kaggle.com/datasets/blastchar/telco-customer-churn)

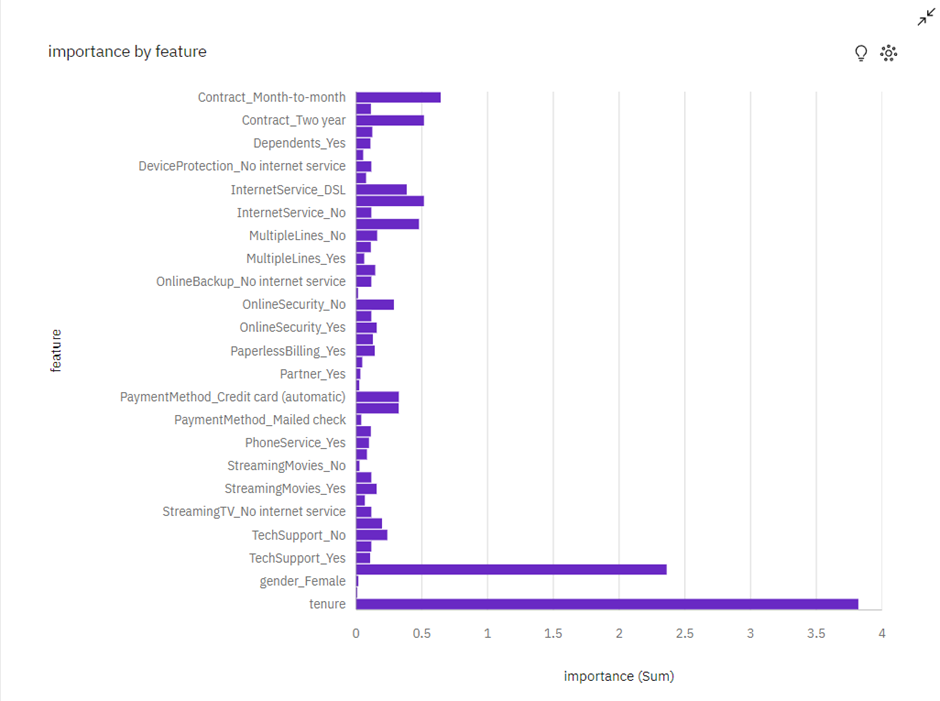
**Synopsis:**

In this phase, models such as Logistic Regression, Random Forest and XGBoost were used to predict which category is most important when it comes to churn prediction.

**Cognos Analytics:**

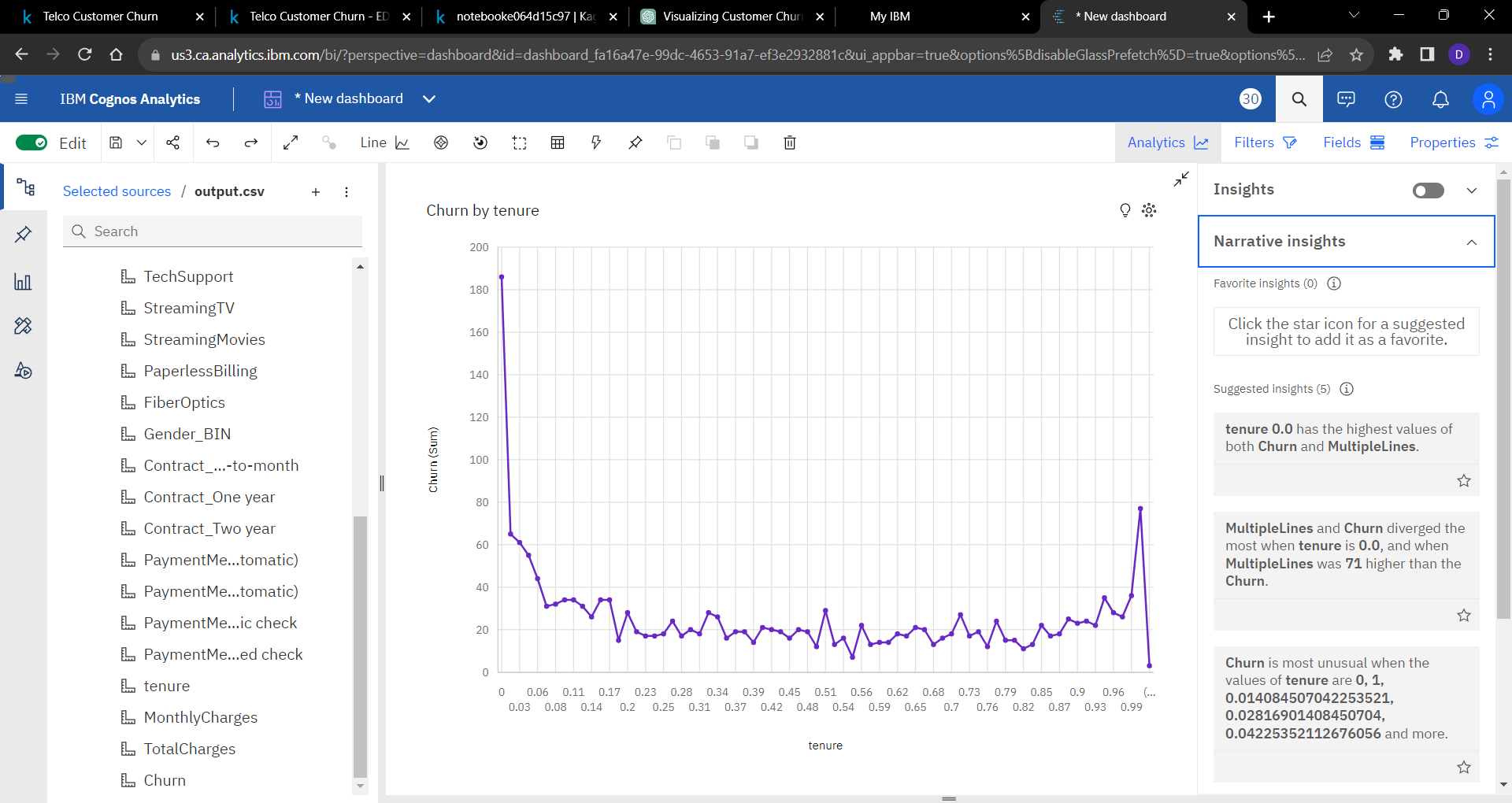
**Logistic Regression:**

Importance is unusually high when feature is tenure and Total\_Charges.feature gender\_Male has the lowest total importance at 0.0, followed by OnlineBackup\_Yes at 0.02. Over all features, the sum of importance is 13.13.For importance, the most significant values of feature are tenure and Total Charges, whose respective importance values add up to 6.181, or 47.1 % of the total.



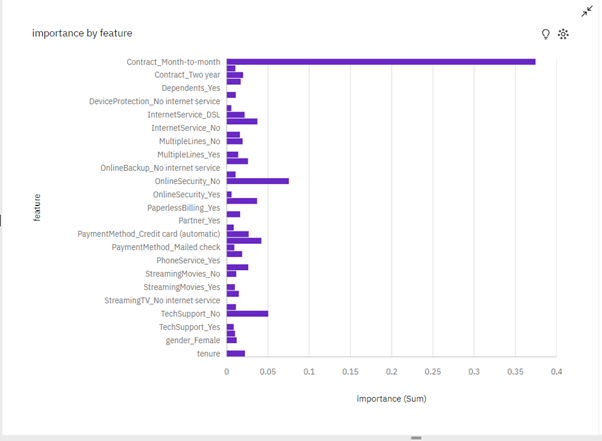
It is clear that the tenure feature has the highest significance in predicting the churn rate. The longer the tenure, the less likely the customer is to churn, as shown by the negative coefficient. The second most significant feature is the Total\_Charges feature with a positive coefficient followed by a positive coefficient of Month-to-month contract and so on.

A line graph was plotted between tenure and churn:



**XGBoost Classifier:**

Importance is unusually high when feature is Contract\_Month-to-month. Feature Dependents\_Yes and deviceprotection\_No internet service have the lowest total importance at 0.0. Feature Contract\_Month-to-month has the highest total importance at 0.37, followed by onlinesecurity\_No at 0.08. Over all features, the sum of importance is 1. Importance ranges from 0, when feature is Dependents\_Yes, to 0.3744, when feature is Contract\_Month-to-month.



Contract\_Month-to-Month has the highest feature importance. This is because the customers who have a month-to-month contract are more likely to churn as compared to customers who have a one-year or two-year contract, since it is easier to terminate a contract on a monthly basis. This is in stark contrast to the Logistic Regression model where the feature importance of Contract\_Month-to-Month was much lower and the feature importance of tenure was much higher.

**Predictive Model:**

data\_dummies = pd.get\_dummies(data)

data\_dummies

X = data\_dummies.drop('Churn', axis=1)

y = data\_dummies['Churn']

features = X.columns.values

scaler = MinMaxScaler(feature\_range = (0,1))

scaler.fit(X)

X = pd.DataFrame(scaler.transform(X))

X.columns = features

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,test\_size=0.3, random\_state=123)

y\_train.value\_counts()

Training on Imbalanced Training Data and Testing on Imbalanced Testing Data

1. Logistic Regression

log\_reg = LogisticRegression()

result = log\_reg.fit(X\_train, y\_train)

y\_preds = log\_reg.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

# Do gridsearch across 10 folds for Logistic Regression

log\_reg = LogisticRegression()

param\_grid = {

            'C': np.logspace(-4, 4, 50),

            'penalty': ['l2', 'none'],

            'solver': ['lbfgs', 'sag', 'saga']

            }

log\_reg\_cv = GridSearchCV(log\_reg, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

log\_reg\_cv.fit(X\_train, y\_train)

print("Tuned Logistic Regression Parameters: {}".format(log\_reg\_cv.best\_params\_))

print("Best score is {}\n\n".format(log\_reg\_cv.best\_score\_))

y\_preds = log\_reg\_cv.best\_estimator\_.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

1. Random Forest

rf = RandomForestClassifier()

result = rf.fit(X\_train, y\_train)

y\_preds = rf.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

y\_preds = rf.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

# Tuning Hyperparameters of RF

rf = RandomForestClassifier()

param\_grid = {

            'n\_estimators': [100, 200, 300, 400, 500],

            'max\_depth': [3, 4, 5, 6, 7, 8, 9, 10],

            'criterion' :['gini', 'entropy']

            }

rf\_cv = GridSearchCV(rf, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

rf\_cv.fit(X\_train, y\_train)

print("Tuned Random Forest Parameters: {}".format(rf\_cv.best\_params\_))

print("Best score is {}\n\n".format(rf\_cv.best\_score\_))

y\_preds = rf\_cv.best\_estimator\_.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

1. XGBoost

xgb = XGBClassifier()

result = xgb.fit(X\_train, y\_train)

y\_preds = xgb.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

# Tuning Hyperparameters of XGB

xgb = XGBClassifier()

param\_grid = {

            'n\_estimators': [100, 200, 300, 400, 500],

            'max\_depth': [3, 4, 5, 6, 7, 8, 9, 10],

            'learning\_rate': [0.01, 0.05, 0.1, 0.15, 0.2]

            }

xgb\_cv = GridSearchCV(xgb, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

xgb\_cv.fit(X\_train, y\_train)

print("Tuned XGB Parameters: {}".format(xgb\_cv.best\_params\_))

print("Best score is {}\n\n".format(xgb\_cv.best\_score\_))

y\_preds = xgb\_cv.best\_estimator\_.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

XGBoost gives more consistent performance on the test data as compared to Logistic Regression and Random Forest indicating that it is a better model for this dataset

**Future Work:**

1. SMOTE analysis: To reduce the class imbalance in the dataset  which will need to be fixed using Oversampling or Undersampling.
2. Feature Engineering and Selection: Using RFE for feature selection. This selects the top 15 features which are most relevant to the model.
3. Model Ensembling: Used a voting classifier with XGBoost and Random Forest, and employed soft voting to give predictions based on the probability scores of the two models.
4. Post-Processing Thresholds: Instead of using a default threshold of 0.5, this adjusts the threshold for classification based on the F1-score, which seeks to balance precision and recall.
5. Regularization: Included the reg\_lambda hyperparameter which is L2 regularization term on weights. The code now includes model ensembling and uses a more sophisticated mechanism for post-processing to adjust the classification threshold. This, in combination with previous enhancements, should further improve the model's performance.

**Conclusion:**

Thus the predictive models were built and important features influencing churn were found and different visualizations were performed using IBM cognos.